Diving into PrestaShop Core development

Table of contents

Diving into PrestaShop Core development

Accessing the database

The database structure

PrestaShop's database tables start with the `ps_` prefix. Note that this can be customized during installation

All table names are in lowercase, and words are separated with an underscore character ("_").

When a table establishes the links between two entities, the names of both entities are mentioned in the table's name. For instance, `ps_category_product` links products to their category.

A few details to note:

- Use the `id_lang` field to store the language associated with a record.
- Use the `id_shop` field to store the store associated with a record.
- Tables which contain translations must end with the `_lang` suffix. For instance, `ps_product_lang` contains all the translations for the `ps_product` table.
- Tables which contain the records linking to a specific shop must end with the `_shop` suffix. For instance, `ps_category_shop` contains the position of each category depending on the store.

The ObjectModel class

This is the main object of PrestaShop's object model. It can be overridden with precaution.

It is an Active Record kind of class (see: http://en.wikipedia.org/wiki/Active_record_pattern). PrestaShop's database table attributes or view attributes are encapsulated in the class. Therefore, the class is tied to a database record. After the object has been instantiated, a new record is added to the database. Each object retrieves its data from the database; when an object is updated, the record to which it is tied is updated as well. The class implements accessors for each attribute.

Defining the model

You must use the `$definition` static variable in order to define the model.

For instance:

```
/**
* Example from the CMS model (CMSCore)
*/
public static $definition = array(
  'table' => 'cms',
  'primary' => 'id_cms',
  'multilang' => true,
  'fields' => array(
    'id_cms_category'  => array('type' => self::TYPE_INT, 'validate' => 'isUnsignedInt'),
    'position'         => array('type' => self::TYPE_INT),
    'active'           => array('type' => self::TYPE_BOOL),
    // Lang fields
    'meta_description' =>
        array('type' => self::TYPE_STRING, 'lang' => true, 'validate' => 'isGenericName', 'size' => 255),
    'meta_keywords'    =>
        array('type' => self::TYPE_STRING, 'lang' => true, 'validate' => 'isGenericName', 'size' => 255),
    'meta_title'       =>
        array('type' => self::TYPE_STRING, 'lang' => true, 'validate' => 'isGenericName', 'required' => true,
'size' => 128),
    'link_rewrite'     =>
        array('type' => self::TYPE_STRING, 'lang' => true, 'validate' => 'isLinkRewrite', 'required' => true,
'size' => 128),
    'content'          =>
        array('type' => self::TYPE_HTML,   'lang' => true, 'validate' => 'isString', 'size' => 3999999999999),
  ),
);
```

A model for many stores and/or languages

In order to have an object in many languages:

```
'multilang' => true
```

In order to have an object depending on the current store

```
'multishop' => true
```

In order to have an object which depends on the current store, and in many languages:

```
'multilang_shop' => true
```

The main methods

Any overriding of the ObjectModel methods is bound to influence how all the other classes and methods act. Use with care.

Method name and parameters	Description
__construct($id = NULL, $id_lang = NULL)	Build object.
add($autodate = true, $nullValues = false)	Save current object to database (add or update).
associateTo(integer|array $id_shops)	Associate an item to its context.
delete()	Delete current object from database.

deleteImage(mixed $force_delete = false)	Delete images associated with the object.
deleteSelection($selection)	Delete several objects from database.
getFields()	Prepare fields for ObjectModel class (add, update).
getValidationRules($className = _CLASS_)	Return object validation rules (field validity).
save($nullValues = false, $autodate = true)	Save current object to database (add or update).
toggleStatus()	Toggle object's status in database.
update($nullValues = false)	Update current object to database.
validateFields($die = true, $errorReturn = false)	Check for field validity before database interaction.

The DBQuery class

The DBQuery class is a query builder which helps you create SQL queries. For instance:

```
$sql = new DbQuery();
$sql->select('*');
$sql->from('cms', 'c');
$sql->innerJoin('cms_lang', 'l', 'c.id_cms = l.id_cms AND l.id_lang = '.(int)$id_lang);
$sql->where('c.active = 1');
$sql->orderBy('position');
return Db::getInstance()->executeS($sql);
```

Here are some of the methods from this class:

Method name and parameters	Description
__toString()	Generate and get the query.
build()	Generate and get the query (return a string).
from(string $table, mixed $alias = null)	Set table for FROM clause.
groupBy(string $fields)	Add a GROUP BY restriction.
having(string $restriction)	Add a restriction in the HAVING clause (each restriction will be separated by an AND statement).
innerJoin(string $table, string $alias = null, string $on = null)	Add INNER JOIN clause, E.g. $this->innerJoin('product p ON ...').
join(string $join)	Add JOIN clause, E.g. $this->join('RIGHT JOIN'. _DB_PREFIX_.'product p ON ...');.
leftJoin(string $table, string $alias = null, string $on = null)	Add LEFT JOIN clause.
leftOuterJoin(string $table, string $alias = null, string $on = null)	Add LEFT OUTER JOIN clause.
limit(string $limit, mixed $offset = 0)	Limit results in query.
naturalJoin(string $table, string $alias = null)	Add NATURAL JOIN clause.
orderBy(string $fields)	Add an ORDER B restriction.
select(string $fields)	Add fields in query selection.
where(string $restriction)	Add a restriction in WHERE clause (each restriction will be separated by an AND statement).

The Dispatcher

The Dispatcher is one of the main technical features of v1.5. It handles URL redirections. Instead of using multiple files in the root folder like `product.php`, `order.php` or `category.php`, only one file is used: `index.php`. From now on, internal URL will look like `index.php?controller=category`, `index.php?controller=product`, etc.

Additionally, the Dispatcher is built to support URL rewriting. Therefore, when URL-rewriting is off, PrestaShop will use the following URL form:

```
http://myprestashop.com/index.php?controller=category&id_category=3&id_lang=1
http://myprestashop.com/index.php?controller=product&id_product=1&id_lang=2
```

...and when URL-rewriting is on (or "Friendly URLs"), PrestaShop's Dispatcher will correctly support this URL form:

```
http://myprestashop.com/en/3-music-ipods
http://myprestashop.com/fr/1-ipod-nano.html
```

There are several advantages for this system:

- It is easier to add a controller.
- You can use custom routes to change your friendly URLs (which is really better for SEO!)
- There is only one single entry point into the software, which improves PrestaShop's reliability, and facilitates future developments.

The Dispatcher makes use of three new 1.5 abstract classes: `Controller`, `FrontController` and `AdminController` (the last two inheriting from the first one).

New routes can be created by overriding the `loadRoutes()` method.
The store administrator can change a controller's URL using the "SEO & URLs" page in the back-office's "Preferences" menu.

Controllers

In the MVC architecture, a Controller manages the synchronization events between the View and the Model, and keeps them up to date. It receives all the user events and triggers the actions to perform. If an action needs data to be changed, the Controller will "ask" the Model to change the data, and in turn the Model will notify the View that the data has been changed, so that the View can update itself.

All of PrestaShop's controllers actually override the `Controller` class through another inheriting class, such as `AdminController`, `ModuleAdminController`, `FrontController`, `ModuleFrontController`, etc.

The FrontController class

Some of the class' properties:

Property	Description
$template	Template name for page content.
$css_files	Array list of CSS files.
$js_files	Array list of JavaScript files.
$errors	Array of errors that have occurred.

$guestAllowed	Whether a customer who has signed out can access the page.
$initialized	Whether the `init()` function has been called.
$iso	The ISO code of the currently selected language.
$n	The number of items per page.
$orderBy	The field used to sort.
$orderWay	Whether to sort is ascending or descending ("ASC" or "DESC").
$p	The current page number.
$ajax	If the ajax parameter is detected in request, set this flag to true.

Execution order of the controller's functions

1. `__contruct()`: Sets all the controller's member variables.
2. `init()`: Initializes the controller.
3. `setMedia()` or `setMobileMedia()`: Adds all JavaScript and CSS specifics to the page so that they can be combined, compressed and cached (see PrestaShop's CCC tool, in the back-office "Performance" page, under the "Advanced preferences" menu).
4. `postProcess()`: Handles `ajaxProcess`.
5. `initHeader()`: Called before `initContent()`.
6. `initContent()`: Initializes the content.
7. `initFooter()`: Called after `initContent()`.
8. `display()` or `displayAjax()`: Displays the content.

Existing controllers

Controller's filename	Description
AddressController.php	Used by address.php to edit a customer's address.
AddressesController.php	Used by addresses.php to get customer's addresses.
AuthController.php	Used by authentication.php for customer login.
BestSalesController.php	Used by best-sales.php to get best-sellers.
CartController.php	Used by cart.php to manage the customer's cart.
CategoryController	Used by category.php to get product categories.
CMSController.php	Used by cms.php to get a CMS page.
CompareController.php	Used by products-comparison.php to compare products.
ContactController.php	Used by contact-form.php to send messages.
DiscountController.php	Used by discount.php to get a customer's vouchers.
GuestTrackingController.php	Used by guest-tracking.php to manage guest orders.
HistoryController.php	Used by history.php to get a customer's orders.
IdentityController.php	Used by identity.php for customer's personal info.
IndexController.php	Used by index.php to display the homepage.
ManufacturerController.php	Used by manufacturer.php to get manufacturers.
MyAccountController.php	Used by my-account.php to manage customer account.
NewProductsController.php	Used by new-products.php to get new products.
OrderConfirmationController.php	Used by order-confirmation.php for order confirmation.
OrderController.php	Used by order.php to manage the five-step checkout.
OrderDetailController.php	Used by order-detail.php to get a customer order.
OrderFollowController.php	Used by order-follow.php to get a customer's returns.
OrderOpcController.php	Used by order-opc.php to manage one-page checkout.
OrderReturnController.php	Used by order-return.php to get a merchandise return.

OrderSlipController.php	Used by order-slip.php to get a customer's credit slips.
PageNotFoundController.php	Used by 404.php to manage the "Page not found" page.
ParentOrderController.php	Manages shared order code.
PasswordController.php	Used by password.php to reset a lost password.
PricesDropController.php	Used by prices-drop.php to get discounted products.
ProductController.php	Used by product.php to get a product.
SearchController.php	Used by search.php to get search results.
SitemapController.php	Used by sitemap.php to get the sitemap.
StoresController.php	Used by stores.php to get store information.
SupplierController.php	Used by supplier.php to get suppliers.

Overriding a controller

Thanks to object inheritance, you can change a controller's behaviors, or add new ones.

PrestaShop's controllers are all stored in the `/controllers` folder, and use the "Core" suffix.

For instance, when working with the Category controller:

- File: `/controllers/CategoryController.php`
- Class: `CategoryControllerCore`

In order to override a controller, you must first create a new class without the "Core" suffix, and place its file in the `/override/controllers` folder.

For instance, when overriding the Category controller:

- File: `/override/controllers/front/CategoryController.php`
- Class: `CategoryController`

Views

PrestaShop uses the Smarty template engine to generate its views: http://www.smarty.net/

The views are stored in `.tpl` files.

A view name is generally the same as the name for the code using it. For instance, `404.php` uses `404.tpl`.

View overriding

As there is no inheritance, there is no way to override a view.

In order to change a view, you must rewrite the template file, and place it in your theme's folder.

Cookies

PrestaShop uses encrypted cookies to store all the session information, for visitors/clients as well as for employees/administrators.

The Cookie class (`/classes/Cookie.php`) is used to read and write cookies.

In order to access the cookies from within PrestaShop code, you can use this:

```
$this->context->cookie;
```

All the information stored within a cookie is available using this code:

```
$this->context->cookie->variable;
```

If you need to access the PrestaShop cookie from non-PrestaShop code, you can use this code:

```
include_once('path_to_prestashop/config/config.inc.php');
include_once('path_to_prestashop/config/settings.inc.php');
include_once('path_to_prestashop/classes/Cookie.php');
$cookie = new Cookie('ps'); // Use "psAdmin" to read an employee's cookie.
```

Data stored in a visitor/client's cookie

Token	Description
date_add	The date and time the cookie was created (in YYYY-MM-DD HH:MM:SS format).
id_lang	The ID of the selected language.
id_currency	The ID of the selected currency.
last_visited_category	The ID of the last visited category of product listings.
ajax_blockcart_display	Whether the cart block is "expanded" or "collapsed".
viewed	The IDs of recently viewed products as a comma-separated list.
id_wishlist	The ID of the current wishlist displayed in the wishlist block.
checkedTOS	Whether the "Terms of service" checkbox has been ticked (1 if it has and 0 if it hasn't)
id_guest	The guest ID of the visitor when not logged in.
id_connections	The connection ID of the visitor's current session.
id_customer	The customer ID of the visitor when logged in.
customer_lastname	The last name of the customer.
customer_firstname	The first name of the customer.
logged	Whether the customer is logged in.
passwd	The MD5 hash of the `_COOKIE_KEY_` in `config/settings.inc.php` and the password the customer used to log in.
email	The email address that the customer used to log in.
id_cart	The ID of the current cart displayed in the cart block.
checksum	The Blowfish checksum used to determine whether the cookie has been modified by a third party. The customer will be logged out and the cookie deleted if the checksum doesn't match.

Data stored in an employee/administrator's cookie

Token	Description
date_add	The date and time the cookie was created (in YYYY-MM-DD HH:MM:SS format).
id_lang	The ID of the selected language.
id_employee	The ID of the employee.
lastname	The last name of the employee.
firstname	The first name of the employee.

email	The email address the employee used to log in.
profile	The ID of the profile that determines which tabs the employee can access.
passwd	The MD5 hash of the `_COOKIE_KEY_` in `config/settings.inc.php` and the password the employee used to log in.
checksum	The Blowfish checksum used to determine whether the cookie has been modified by a third party. If the checksum doesn't match, the customer will be logged out and the cookie is deleted .

Hooks

Hooks are a way to associate your code to some specific PrestaShop events.

Most of the time, they are used to insert content in a page.

For instance, the PrestaShop default theme's home page has the following hooks:

Hook name	Description
displayHeader	Displays the content in the page's header area.
displayTop	Displays the content in the page's top area.
displayLeftColumn	Displays the content in the page's left column.
displayHome	Displays the content in the page's central area.
displayRightColumn	Displays the content in the page's right column.
displayFooter	Displays the content in the page's footer area.

Hooks can also be used to perform specific actions under certain circumstances (i.e. sending an e-mail to the client).

You can get a full list of the hooks available in PrestaShop 1.5 in the "Hooks in PrestaShop 1.5" chapter of the Developer Guide.

Using hooks

...in a controller

It is easy to call a hook from within a controller: you simply have to use its name with the `hookExec()` method: `Module::hookExec('NameOfHook');`

For instance:

```
$this->context->smarty->assign('HOOK_LEFT_COLUMN', Module::hookExec('displayLeftColumn'));
```

...in a module

In order to attach your code to a hook, you must create a non-static public method, starting with the "`hook`" keyword followed by either "`display`" or "`action`", and the name of the hook you want to use.

This method receives one (and only one) argument: an array of the contextual information sent to the hook.

```
public function hookDisplayNameOfHook($params)
{
    // Your code.
}
```

In order for a module to respond to a hook call, the hook must be registered within PrestaShop. Hook registration is done using the `registerHook()` method. Registration is usually done during the module's installation.

```
public function install()
{
    return parent::install() && $this->registerHook('NameOfHook');
}
```

...in a theme

It is easy to call a hook from within a template file (`.tpl`): you simply have to use its name with the `hool` function. You can add the name of a module that you want the hook execute.

For instance:

```
{hook h='displayLeftColumn' mod='blockcart'}
```

Creating your own hook

You can create new PrestaShop hooks by adding a new record in the `ps_hook` table in your MySQL database. You could do it the hard way:

```
INSERT INTO `ps_hook` (`name`, `title`, `description`) VALUES ('nameOfHook', 'The name of your hook', 'This is a custom hook!');
```

...but PrestaShop enables you to do it the easy way:

```
$this->registerHook('NameOfHook');
```

If the hook "NameOfHook" doesn't exist, PrestaShop will create it for you. No need to do the SQL query anymore.